**《农产品大数据分析系统》**

**详细分析与设计**

**V1.0**

**版 本 历 史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本/状态 | 作者 | 日期 | 备注 |
| 1.0 | 陈嘉、冯文豪、吴明羽、朱明航 | 2020-07-29 | 创建 |
|  |  |  |  |

目录

[第一部分 引言 5](#_Toc48506061)

[一、编写目的 5](#_Toc48506062)

[二、项目背景 5](#_Toc48506063)

[三、定义 5](#_Toc48506064)

[1、VUE 5](#_Toc48506065)

[2、SpringBoot 6](#_Toc48506066)

[3、大数据 6](#_Toc48506067)

[四、参考资料 6](#_Toc48506068)

[第二部分 项目概述 8](#_Toc48506069)

[第三部分 总体设计 9](#_Toc48506070)

[一、技术架构设计 9](#_Toc48506071)

[1、Springboot模型设计 9](#_Toc48506072)

[2、MVVM框架 9](#_Toc48506073)

[3、Axios技术 10](#_Toc48506074)

[第四部分 界面设计 12](#_Toc48506075)

[一、界面框架设计 12](#_Toc48506076)

[二、PC界面设计 12](#_Toc48506077)

[1、主界面： 12](#_Toc48506078)

[2、功能界面 13](#_Toc48506079)

[2.1、抓取量监控和查询界面 13](#_Toc48506080)

[2.2、区域行情界面 13](#_Toc48506081)

[2.3、价格走势界面 14](#_Toc48506082)

[2.4、价格对比界面 14](#_Toc48506083)

[2.5、价格预测界面 15](#_Toc48506084)

[第五部分 单元模块设计 16](#_Toc48506085)

[一、 数据逻辑层 16](#_Toc48506086)

[二、业务逻辑层设计 16](#_Toc48506087)

[1、类图设计 16](#_Toc48506088)

[2、类的详细描述 16](#_Toc48506089)

[三、价格查询模块设计 18](#_Toc48506090)

[1、价格查询 18](#_Toc48506091)

[2、价格对比 23](#_Toc48506092)

[3、区域行情 27](#_Toc48506093)

[四、数据监控模块设计 30](#_Toc48506094)

[1、数据监控 30](#_Toc48506095)

[2、抓取量查询 32](#_Toc48506096)

[五. 接口文档说明 36](#_Toc48506097)

[第六部分 数据库设计 37](#_Toc48506098)

[一、数据库整体结构图 37](#_Toc48506099)

[二、数据库表格清单 37](#_Toc48506100)

[1、Market表结构 37](#_Toc48506101)

[2、Price表结构 38](#_Toc48506102)

[3、Variety表结构 38](#_Toc48506103)

[4、Info\_count表结构 39](#_Toc48506104)

[5、数据库外键清单 39](#_Toc48506105)

[6、数据库建表sql语句 39](#_Toc48506106)

[第七部分 数据爬取设计 46](#_Toc48506107)

[一、数据爬取流程 46](#_Toc48506108)

[二、Spider爬取数据 46](#_Toc48506109)

[1.爬取数据 46](#_Toc48506110)

[2.提高效率 46](#_Toc48506111)

[三、程序类图设计 47](#_Toc48506112)

[第八部分 数据处理设计 49](#_Toc48506113)

[一、数据处理流程 49](#_Toc48506114)

[二、Spark对数据进行处理 49](#_Toc48506115)

[1.数据读取 49](#_Toc48506116)

[2.数据处理 49](#_Toc48506117)

[3.数据存储 50](#_Toc48506118)

[4.分析程序类图设计 50](#_Toc48506119)

第一部分 引言

## 一、编写目的

编写本设计的目的是为了准确阐述基于大数据的农产品批发市场分析系统的具体实现思路和方法，即系统的详细架构和实现逻辑，主要包括程序系统的结构以及各层次中每个程序的设计考虑。预期读者为项目全体成员，包括运行维护和测试人员。

## 二、项目背景

* 系统名称：农产品批发市场大数据分析系统
* 任务提出者：略。
* 开发者：陈嘉、冯文豪、吴明羽、朱明航。
* 用户和运行该程序系统的计算中心：略。

## 三、定义

### 1、VUE

Vue.js是一套构建用户界面的渐进式框架。与其他重量级框架不同的是，Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 完全有能力驱动采用单文件组件和Vue生态系统支持的库开发的复杂单页应用。

Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

Vue.js 自身不是一个全能框架——它只聚焦于视图层。因此它非常容易学习，非常容易与其它库或已有项目整合。另一方面，在与相关工具和支持库一起使用时，Vue.js也能完美地驱动复杂的单页应用。

### 2、SpringBoot

SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的全新开源的轻量级框架。它基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。

### 3、大数据

大数据（Big data）通常用来形容一个公司创造的大量非结构化数据和半结构化数据，这些数据在下载到关系型数据库用于分析时会花费过多时间和金钱。大数据分析常和云计算联系到一起，因为实时的大型数据集分析需要像MapReduce一样的框架来向数十、数百或甚至数千的电脑分配工作。大数据需要特殊的技术，以有效地处理大量的容忍经过时间内的数据。适用于大数据的技术，包括大规模并行处理（MPP）数据库、数据挖掘、分布式文件系统、分布式数据库、云计算平台、互联网和可扩展的存储系统。大数据概念应用到IT操作工具产生的数据中，大数据可以使IT管理软件供应商解决大广泛的业务决策。IT系统、应用和技术基础设施每天每秒都在产生数据。大数据非结构化或者结构数据都代表了“所有用户的行为、服务级别、安全、风险、欺诈行为等更多操作”的绝对记录。

## 四、参考资料

列出有关的参考资料，如：

1. 本项目的经核准的计划任务书或合同、上级机关的批文；

《技术服务合同》

《项目章程》

《里程碑计划》

《开发计划》

《交付物清单》

1. 属于本项目的其他已发表的文件；

《需求说明》

《概要设计》

1. 本文件中各处引用到的文件资料，包括所要用到的软件开发标准。

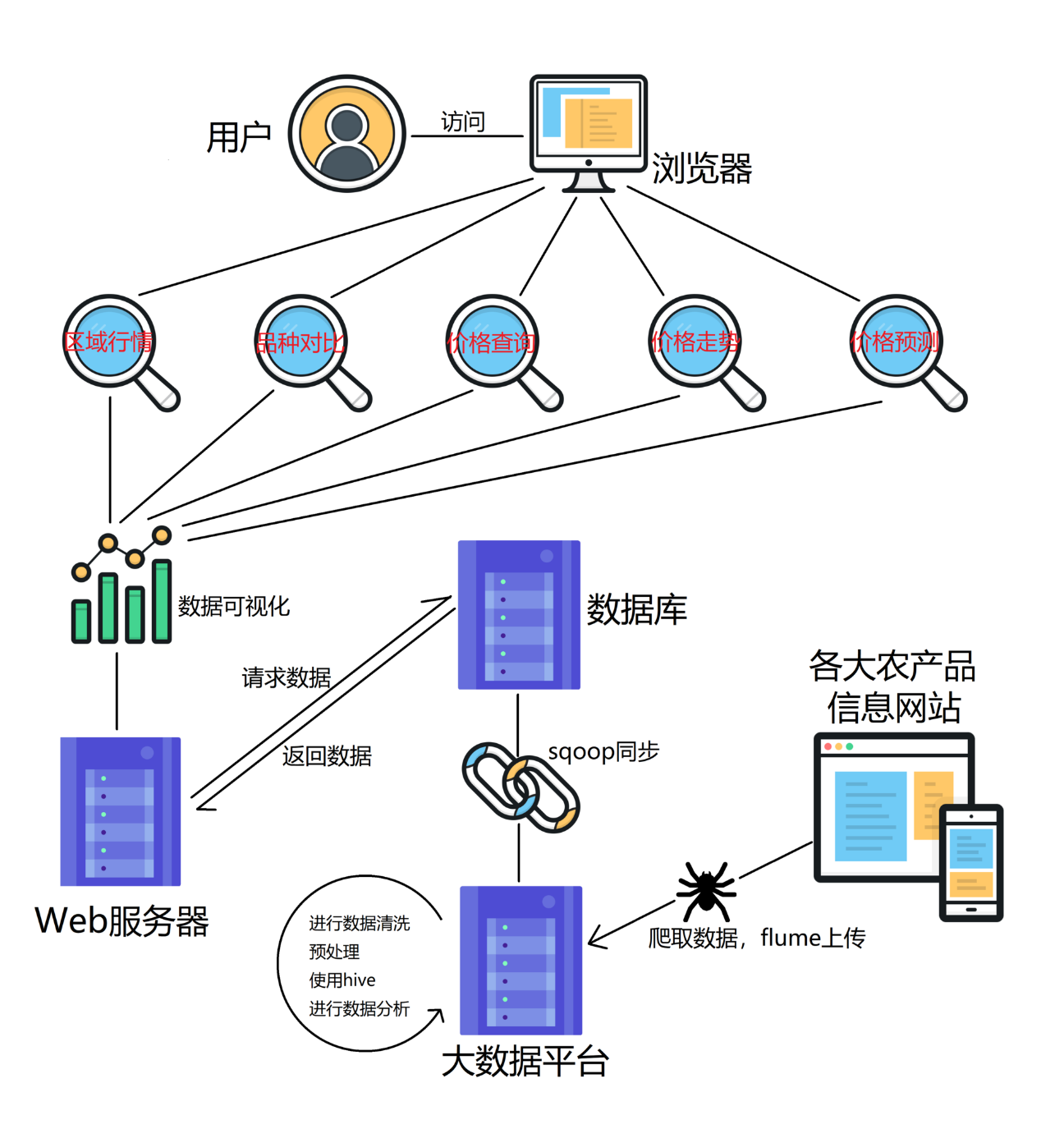
《Java语言编码规范(Java Code Conventions)》

第二部分 项目概述

基于大数据的农产品批发市场分析系统，通过爬虫获取农产品批发市场的各类数据，然后通过大数据平台对数据进行分析处理，最终，将数据可视化的展现给用户查看。

系统主要包括区域行情查询、不同品种间的价格对比、价格走势查询、价格预测、爬取量监测五大部分。

系统可以7\*24小时安全、高效无故障运行，用户可以轻松完成各种查询操作，可以满足用户的各种查询需求。



第三部分 总体设计

## 一、技术架构设计

### 1、Springboot模型设计

系统实现java的SpringBoot系统架构，使用java SDK1.8标准库，使用内嵌的tomcat构建服务器，使用Mybatis Plus对数据库进行操作，使用mysql数据库进行数据的持久化。

1）Controller层

Controller层负责与Web端进行交互，接收Web端发送过来的请求，调用Service层的方法处理请求，最后将后台处理的结果打包发送给Web端。

2）Service层

Service层负责服务数据的处理，实现请求的处理逻辑，调用Mapper层的方法对数据库进行数据的增删改查操作，并将结果返回给Controller层。

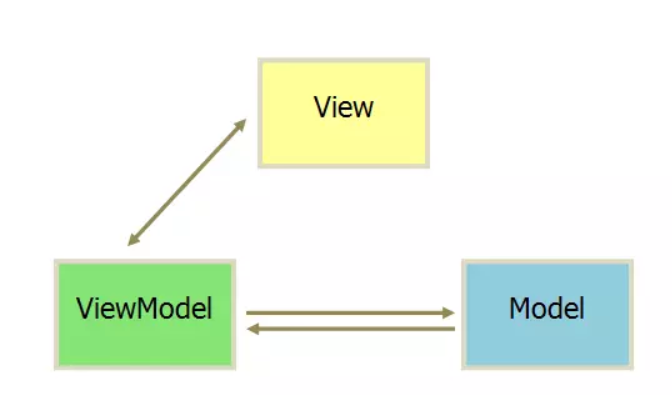
3）Entity层

Entity层负责数据的定义，Entity层中的类规定了数据库中对应的表的结构。

4）Mapper层

Mapper层负责实现与数据库进行交互的方法，对数据库进行操作，将结果返回给Service层。

### 2、MVVM框架



MVVM 由 Model,View,ViewModel 三部分构成，Model 层代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑；View 代表UI 组件，它负责将数据模型转化成UI 展现出来，ViewModel 是一个同步View 和 Model的对象。

在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

### 3、Axios技术

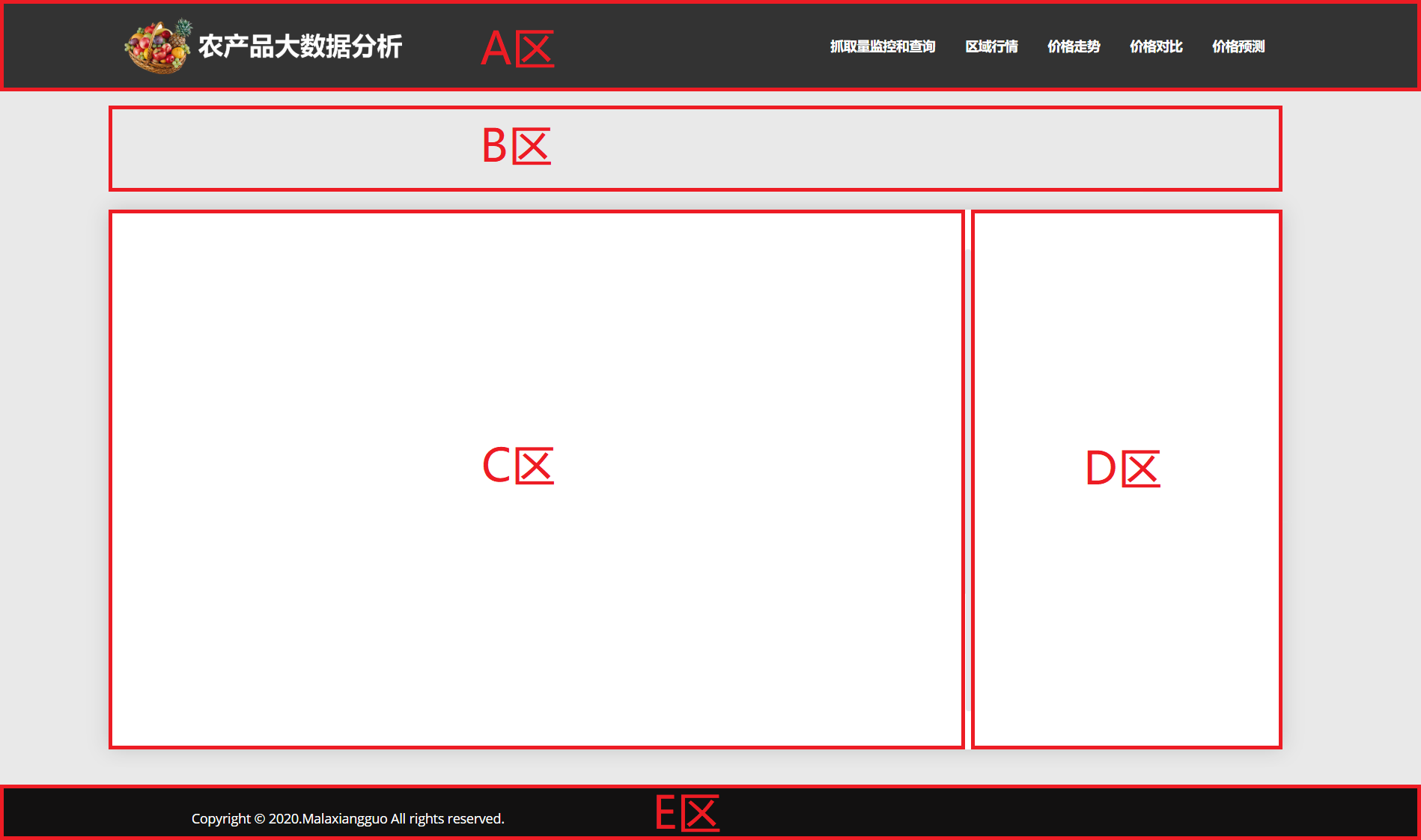
axios主要是用于向后台发起请求的，还有在请求中做更多是可控功能。

axios 特点：

* 从浏览器中创建 XMLHttpRequests
* 从 node.js 创建 http请求
* 支持Promise API
* 拦截请求和响应 （就是有interceptor）
* 转换请求数据和响应数据
* 取消请求
* 自动转换 JSON 数据
* 客户端支持防御

第四部分 界面设计

## 一、界面框架设计



A区：导航栏，由LOGO和其他功能页面的链接组成，起到导航的作用

B区：由一些下拉框和按钮组成，用于用户筛选提交信息

C区：图表展示区，将返回的信息以图表的形式进行展示

D区：展示一些详情信息，以对图标进行补充

## 二、PC界面设计

### 1、主界面：

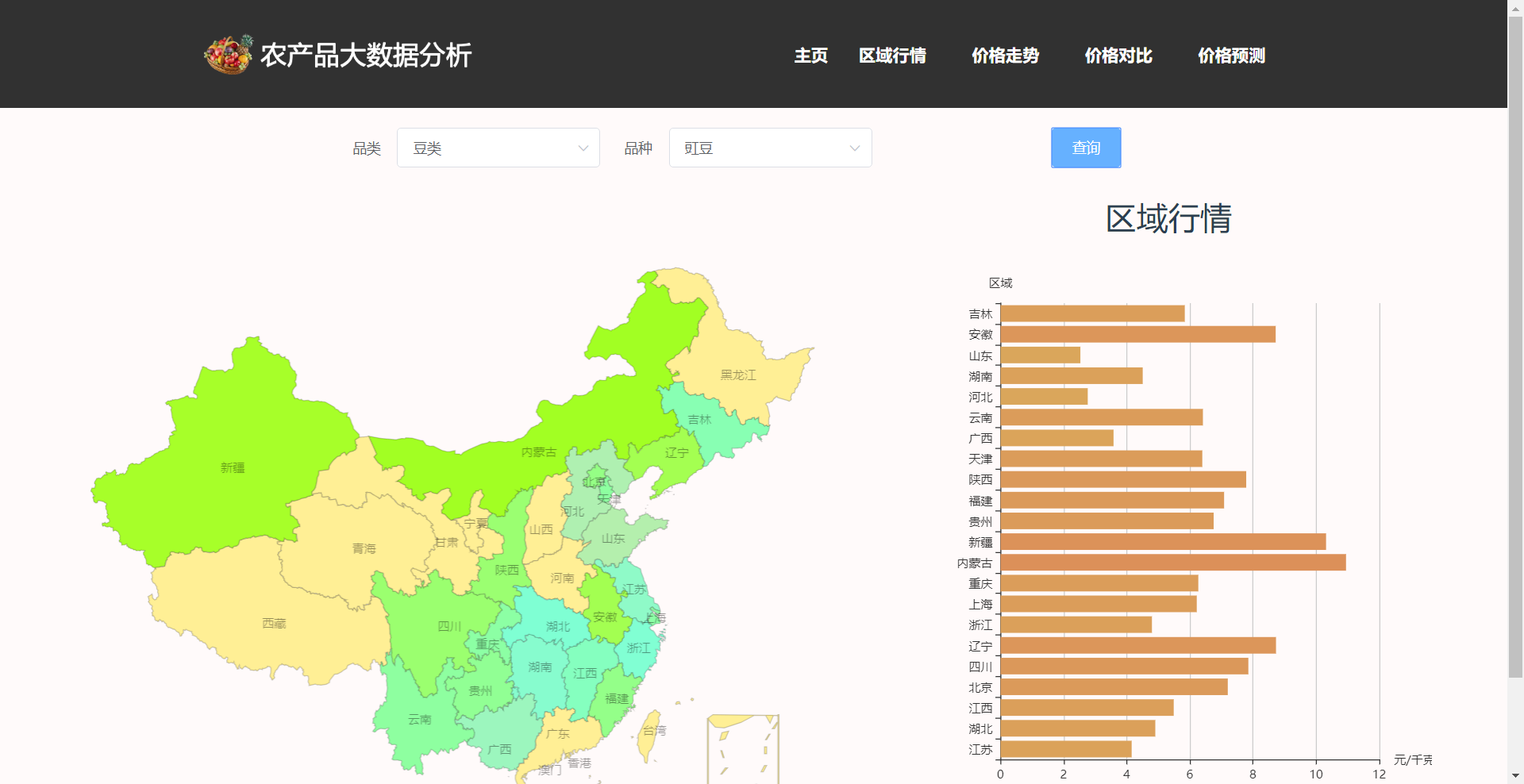


### 2、功能界面

### 2.1、抓取量监控和查询界面



### 2.2、区域行情界面



### 2.3、价格走势界面



### 2.4、价格对比界面

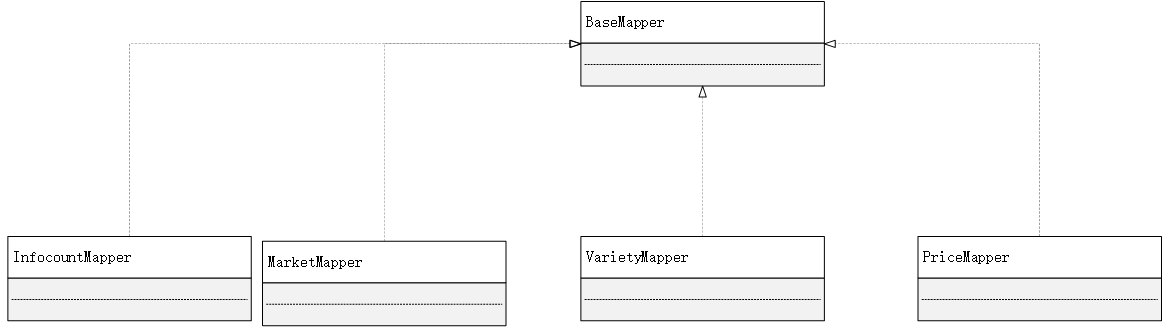


### 2.5、价格预测界面



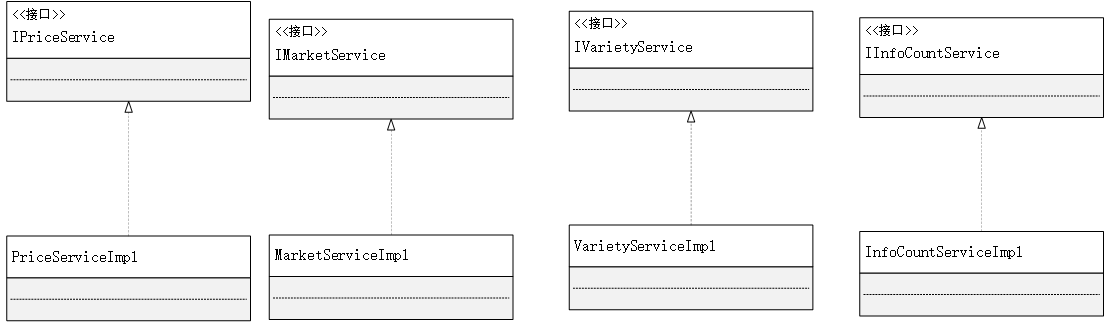
第五部分 单元模块设计

## 一、 数据逻辑层



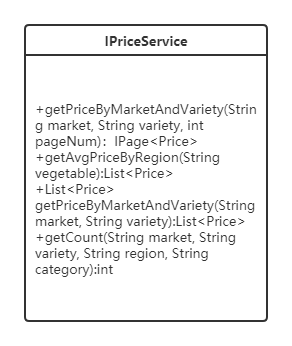
## 二、业务逻辑层设计

### 1、类图设计



### 2、类的详细描述

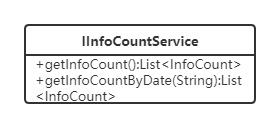
#### 2.1、 IPriceService 接口设计



详细描述

|  |
| --- |
| public IPage<Price> getPriceByMarketAndVariety(String varity, String market,int pageNum)  分页查询指定条件的所有价格列表 |
| public List<Price> getPriceByMarketAndVariety(String varity, String market)  指定条件的所有价格列表 |
| List<Price> getAvgPriceByRegion(String vegetable);  查询某地的平均价格 |
| int getCount(String market, String variety, String region, String category);  按条件查询记录数量 |

#### 2.2、IInfoCountService



详细说明

|  |
| --- |
| public List<Infocount> getInfoCountByDate(String date)  获取一天的实时统计数据 |
| public List<InfoCount>getInfoCount()  获取一段时间的实时统计数据 |

#### 2.3、IMarketService



详细说明

|  |
| --- |
| public List<String> getMarketsByRegion(String region);  查询指定区域内所有批发市场 |
| public List<String> getRegions();  查询所有的批发区域 |

#### 2.4、IVarietyService



详细说明

|  |
| --- |
| public List<String> getCategories();  查询所有的品类 |
| public List<String> getVarietiesByCategory(String);  查询指定品类的所有品种 |

## 三、价格查询模块设计

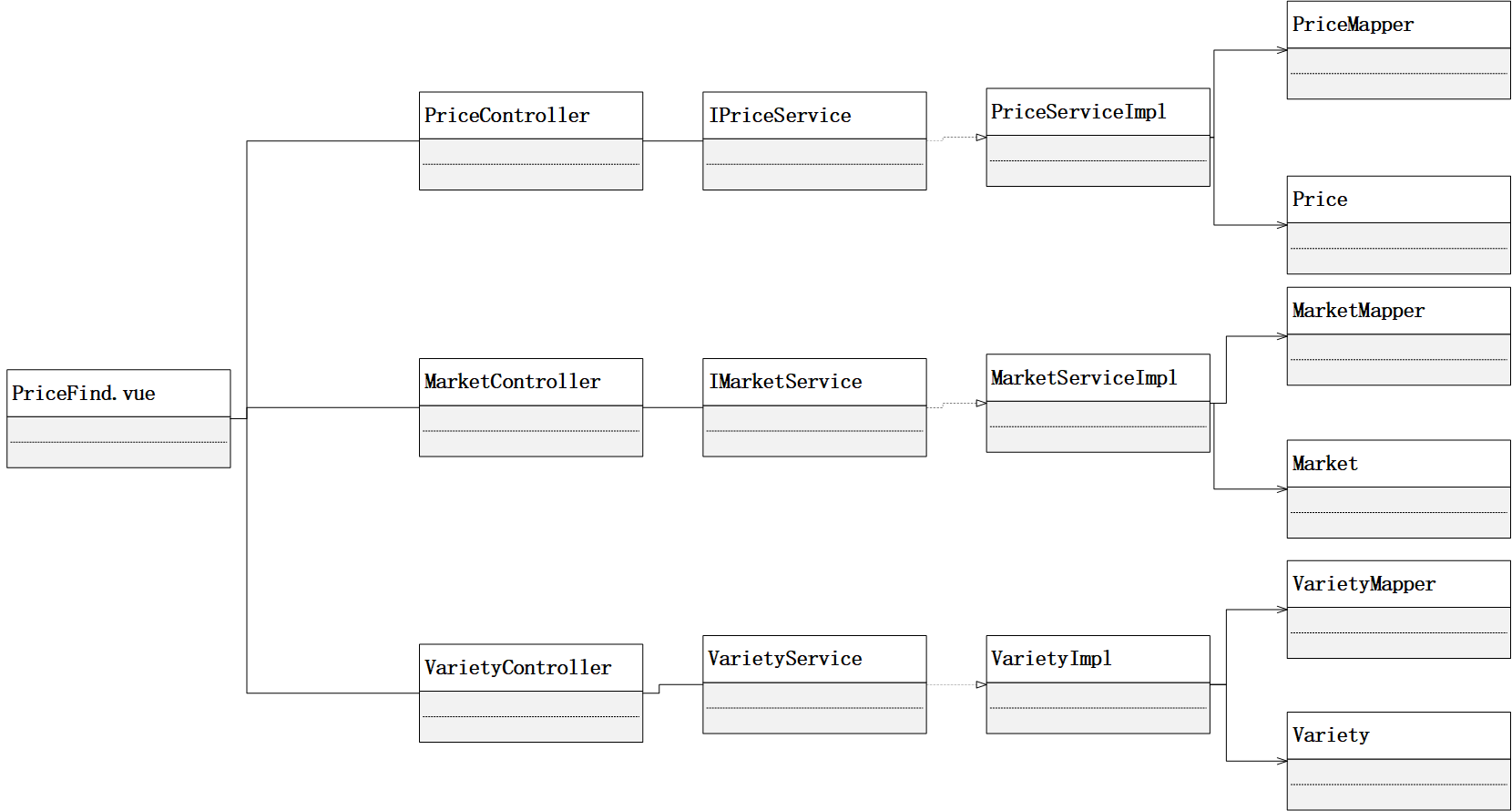
### 1、价格查询

#### 1.1、功能汇总

该功能用户通过输入查询条件包括：区域信息、市场信息、品类信息、品种信息、起止时间信息，提交到服务端进行信息检索，返回查询结果。

|  |  |
| --- | --- |
| 功能 | 描述 |
| 选择查询市场 | 用户在浏览器界面选择查询条件时，根据选择的区域，更改市场下拉列表框的内容 |
| 选择查询品种 | 用户在浏览器界面选择查询条件时，根据选择的品类，更改品种下拉列表框的内容 |
| 显示查询结果 | 用户在提交查询条件，提交至服务端，服务端检索数据库后返回，浏览器显示查询结果 |
| 走势查询 | 点击按钮，将得到的查询结果绘制折线图 |
| 价格对比 | 对所给对比条件分别查询将结果分别处理，分别显示在折线图中 |

#### 1.2、类图设计



#### 1.3、类的详细设计描述

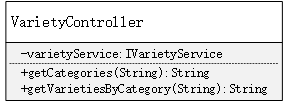
##### 1.3.1、MarketController描述



详细描述：

|  |
| --- |
| MarketController  为用户输入区域和市场提供下拉列表内容 |
| 1.   成员变量  private IMarketService marketService 市场查询服务接口 |
| 2.   public String getRegions() throws Exception  返回包含区域列表数据的Json字符串 |
| 3.   public String getMarketsByRegion(String region) throws Exception  根据输入的区域名返回包含该区域所有市场列表数据的Json字符串 |

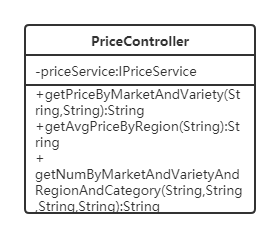
##### 1.3.2、VarietyController描述



详细描述：

|  |
| --- |
| VarietyController  为用户输入品类和品种提供下拉列表内容 |
| 1.   成员变量  private IVarietyService varietyService 市场查询服务接口 |
| 2.   public String getCategories() throws Exception  返回包含品类列表数据的Json字符串 |
| 3.   public String getVarietiessByCategory(String category) throws Exception  根据输入的品类名返回包含该品类所有品种列表数据的Json字符串 |

##### 1.3.3、PriceController描述

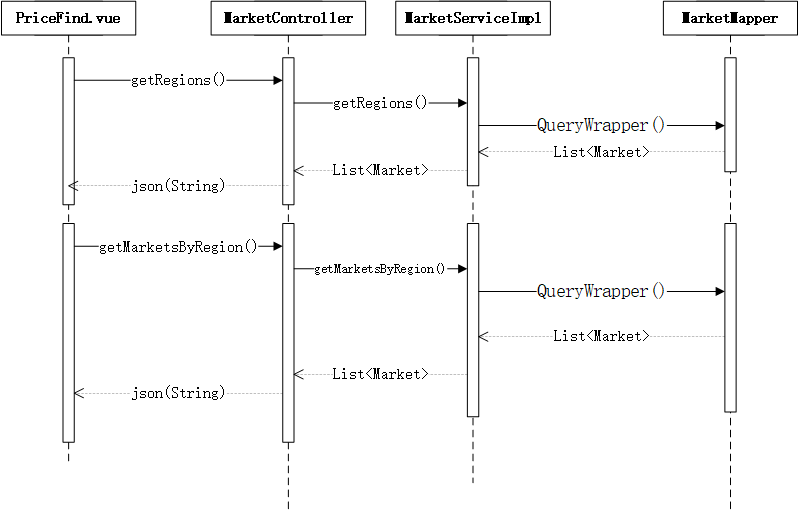


详细描述：

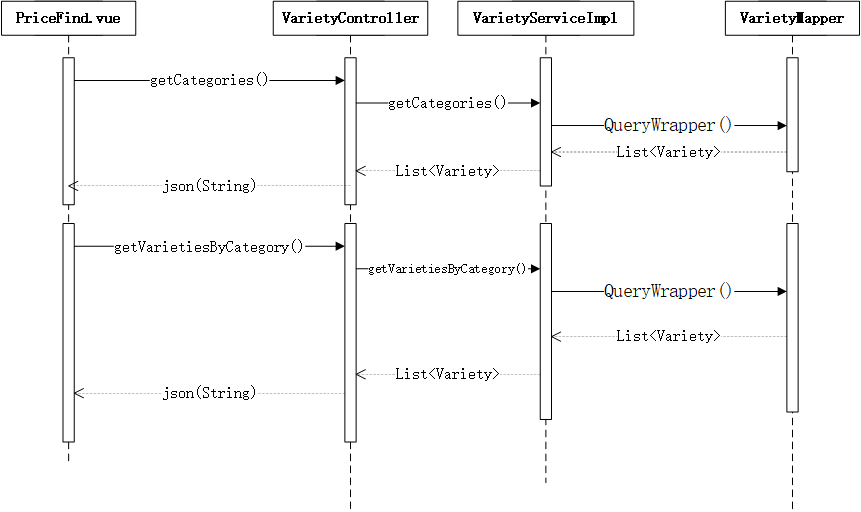
|  |
| --- |
| PriceController  根据用户输入信息执行查询，返回结果 |
| 1.   成员变量  private IPriceService priceService 市场查询服务接口 |
| 2.   public String getPriceByMarketAndVarity(String market, String varity,int pageNum) throws Exception  返回包含执行查询所得结果数据的Json字符串 |
| 3.public String getAvgPriceByRegion(String vegetable)  返回包含执行查询所得结果数据的Json字符串 |
| 4. getNumByMarketAndVarietyAndRegionAndCategory(String market,String variety,String region,String category)public String  返回包含执行查询所得结果数据的Json字符串 |

#### 1.4、功能实现

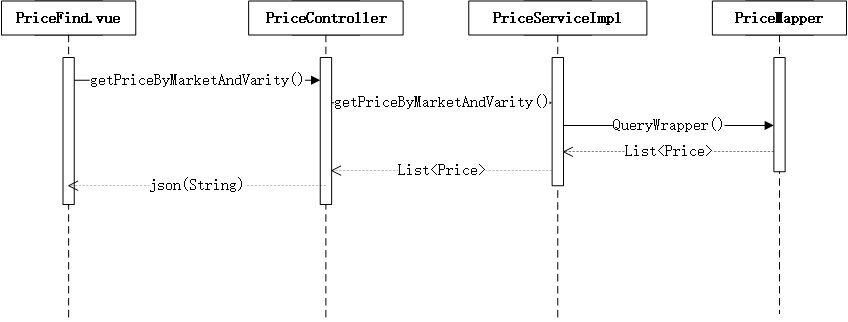
1. 选择查询市场



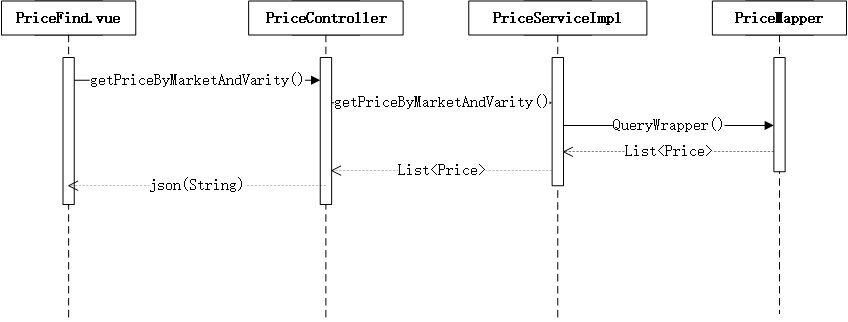
1. 选择查询品种



1. 显示查询结果



1. 走势查询



### 2、价格对比

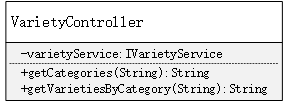
##### 2.3.1、MarketController描述



详细描述：

|  |
| --- |
| MarketController  为用户输入区域和市场提供下拉列表内容 |
| 1.   成员变量  private IMarketService marketService 市场查询服务接口 |
| 2.   public String getRegions() throws Exception  返回包含区域列表数据的Json字符串 |
| 3.   public String getMarketsByRegion(String region) throws Exception  根据输入的区域名返回包含该区域所有市场列表数据的Json字符串 |

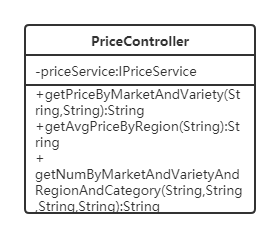
##### 2.3.2、VarietyController描述



详细描述：

|  |
| --- |
| VarietyController  为用户输入品类和品种提供下拉列表内容 |
| 1.   成员变量  private IVarietyService varietyService 市场查询服务接口 |
| 2.   public String getCategories() throws Exception  返回包含品类列表数据的Json字符串 |
| 3.   public String getVarietiessByCategory(String category) throws Exception  根据输入的品类名返回包含该品类所有品种列表数据的Json字符串 |

##### 2.3.3、PriceController描述

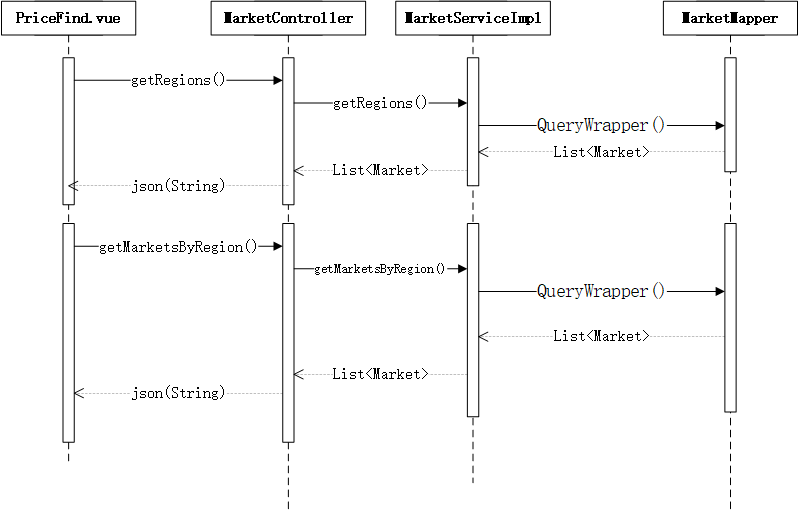


详细描述：

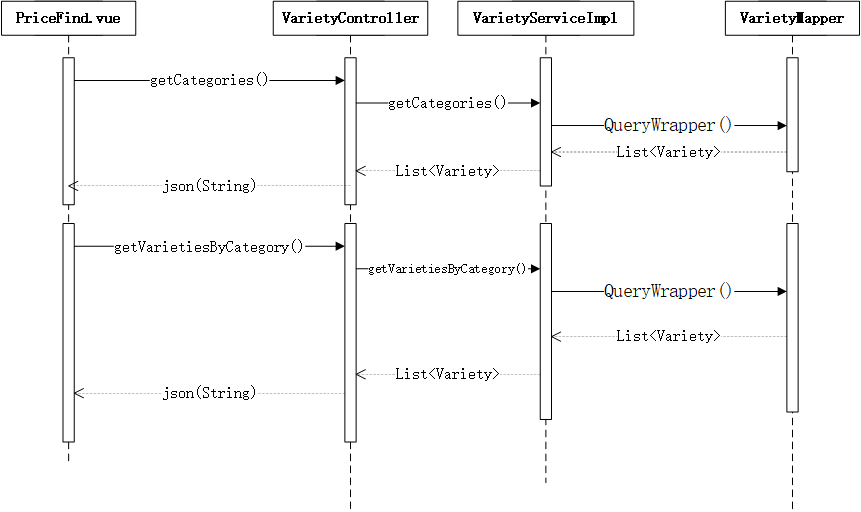
|  |
| --- |
| PriceController  根据用户输入信息执行查询，返回结果 |
| 1.   成员变量  private IPriceService priceService 市场查询服务接口 |
| 2.   public String getPriceByMarketAndVarity(String market, String varity,int pageNum) throws Exception  返回包含执行查询所得结果数据的Json字符串 |
| 3.public String getAvgPriceByRegion(String vegetable)  返回包含执行查询所得结果数据的Json字符串 |
| 4. getNumByMarketAndVarietyAndRegionAndCategory(String market,String variety,String region,String category)public String  返回包含执行查询所得结果数据的Json字符串 |

#### 2.4、功能实现

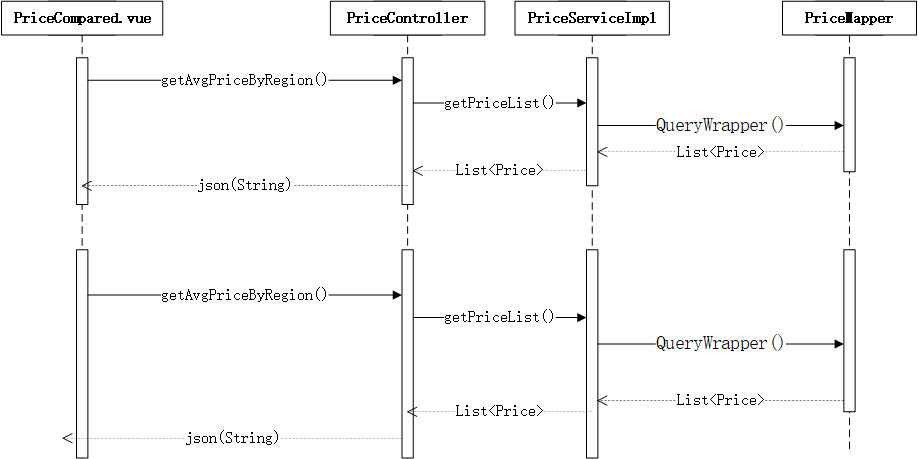
1. 选择查询市场



1. 选择查询品种



1. 重复上述两个步骤
2. 价格对比查询



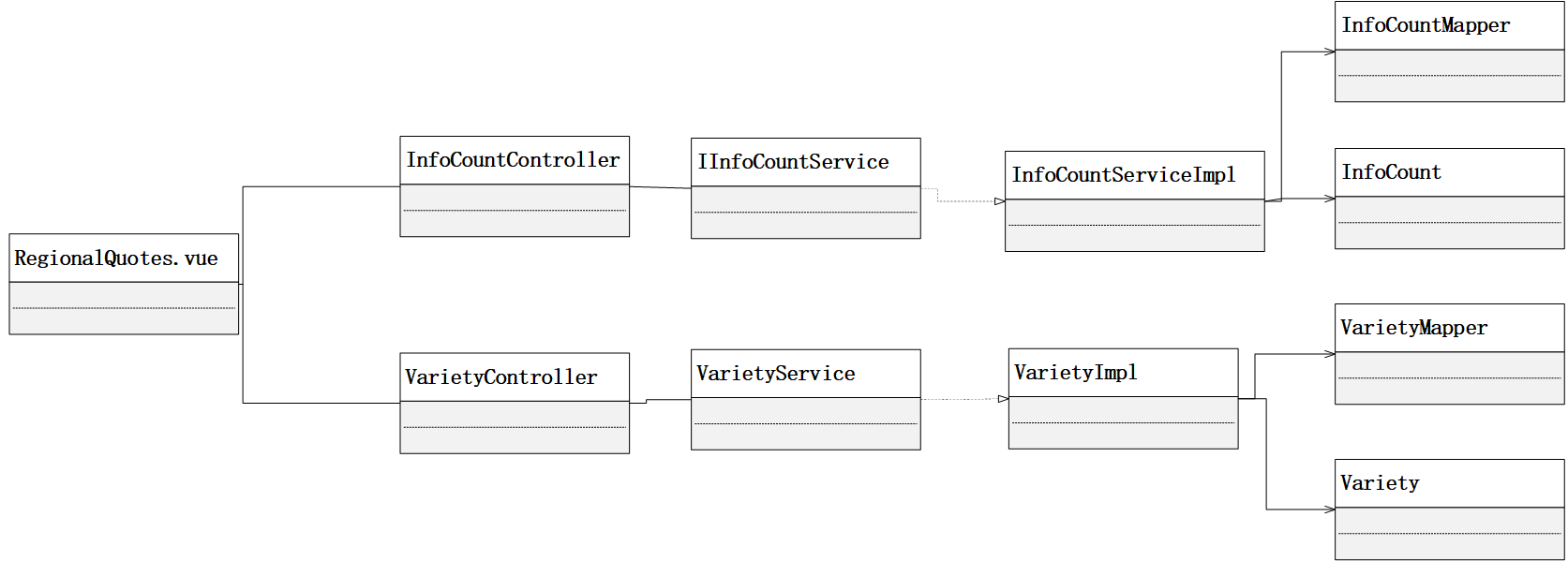
### 3、区域行情

#### 3.1、功能汇总

该功能用户通过输入查询条件包括：品种信息，提交到服务端进行信息检索，返回查询结果。

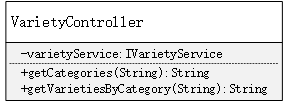
|  |  |
| --- | --- |
| 功能 | 描述 |
| 选择查询品种 | 用户在浏览器界面选择查询条件时，根据选择的品类，更改品种下拉列表框的内容 |
| 区域行情查询 | 用户在提交所要查询的品种，提交至服务端，服务端检索数据库后返回，浏览器显示查询结果 |

#### 3.2、类图设计



#### 3.3、类的详细设计描述

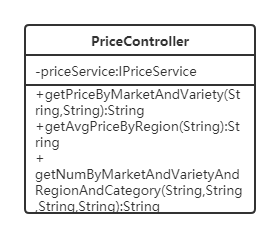
##### 3.3.1、VarietyController描述



详细描述：

|  |
| --- |
| VarietyController  为用户输入品类和品种提供下拉列表内容 |
| 1.   成员变量  private IVarietyService varietyService 市场查询服务接口 |
| 2.   public String getCategories() throws Exception  返回包含品类列表数据的Json字符串 |
| 3.   public String getVarietiesByCategory(String category) throws Exception  根据输入的品类名返回包含该品类所有品种列表数据的Json字符串 |

##### 3.3.2、PriceController描述

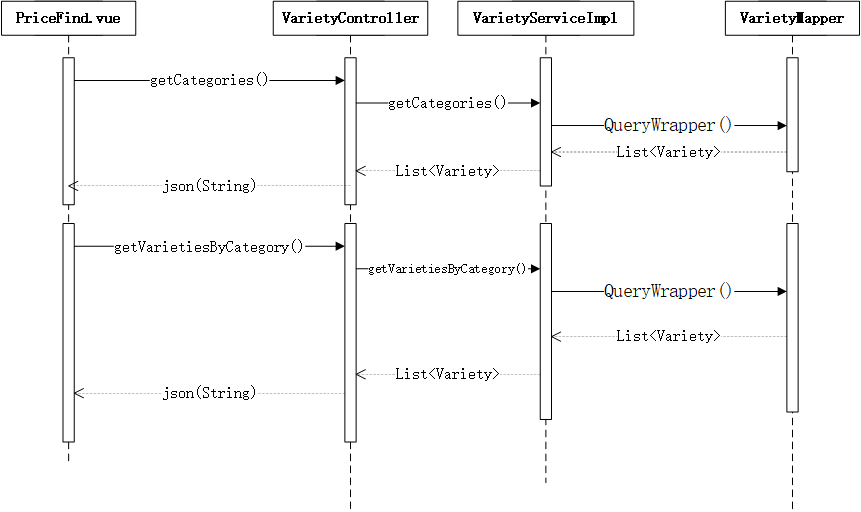


详细描述：

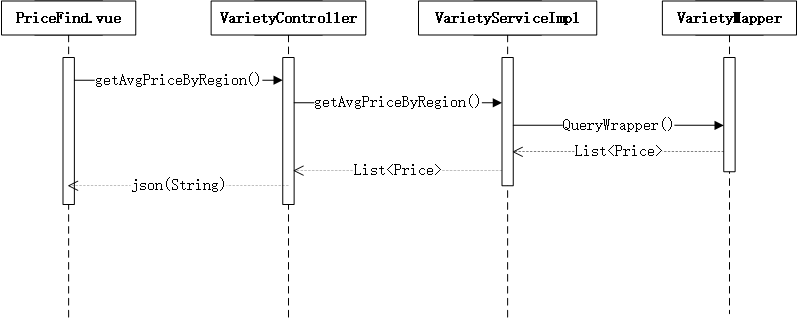
|  |
| --- |
| PriceController  根据用户输入信息执行查询，返回结果 |
| 1.   成员变量  private IPriceService priceService 市场查询服务接口 |
| 2.   public String getPriceByMarketAndVarity(String market, String varity,int pageNum) throws Exception  返回包含执行查询所得结果数据的Json字符串 |
| 3.public String getAvgPriceByRegion(String vegetable)  返回包含执行查询所得结果数据的Json字符串 |
| 4. getNumByMarketAndVarietyAndRegionAndCategory(String market,String variety,String region,String category)public String  返回包含执行查询所得结果数据的Json字符串 |

#### 3.4、功能实现

1. 选择查询品种



1. 区域行情



## 四、数据监控模块设计

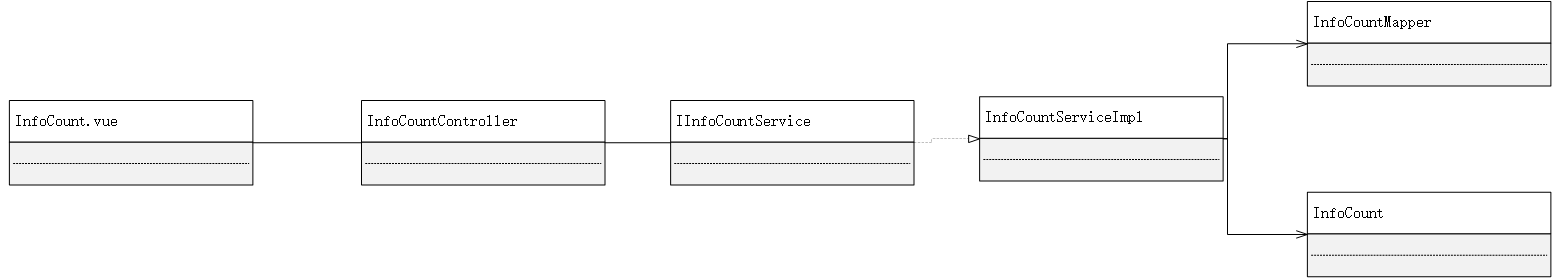
### 1、数据监控

#### 1.1、功能汇总

该功能用户通过输入查询条件包括：区域信息、市场信息、品类信息、品种信息、起止时间信息，提交到服务端进行信息检索，返回查询结果。

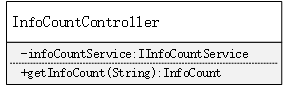
|  |  |
| --- | --- |
| 功能 | 描述 |
| 获取当天数据统计情况 | 用户切换到数据监控界面，向后台查询当天的数据统计情况，前端得到返回结果并显示 |

#### 1.2、类图设计



#### 1.3、类的详细设计描述

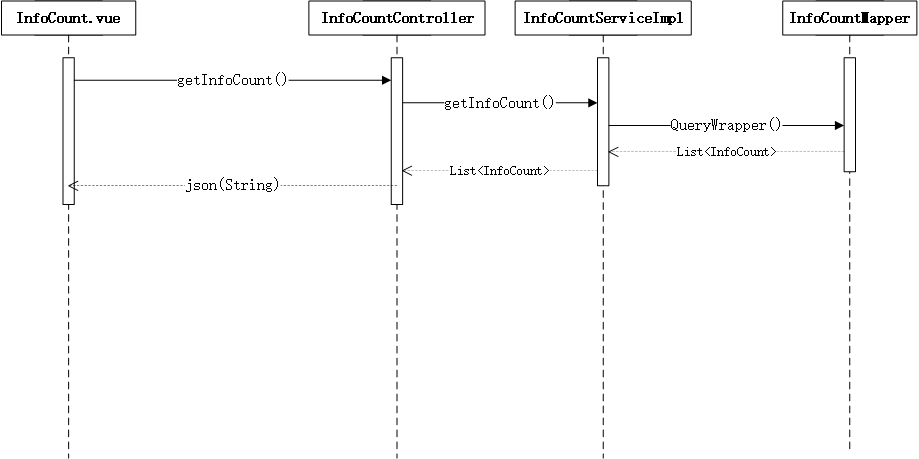
##### 1.3.1 InfoCountController描述



详细描述

|  |
| --- |
| InfoCountController  根据用户输入信息执行查询，返回结果 |
| 1.   成员变量  private IInfoCountService infoCountService 市场查询服务接口 |
| 2.   public int getCountByMarketAndVarity(String market, String varity) throws Exception  返回包含执行查询所得结果数据的总量 |
| 3. public InfoCount getInfoCount(String)  返回某日的所有数据统计信息 |

#### 1.4、功能实现



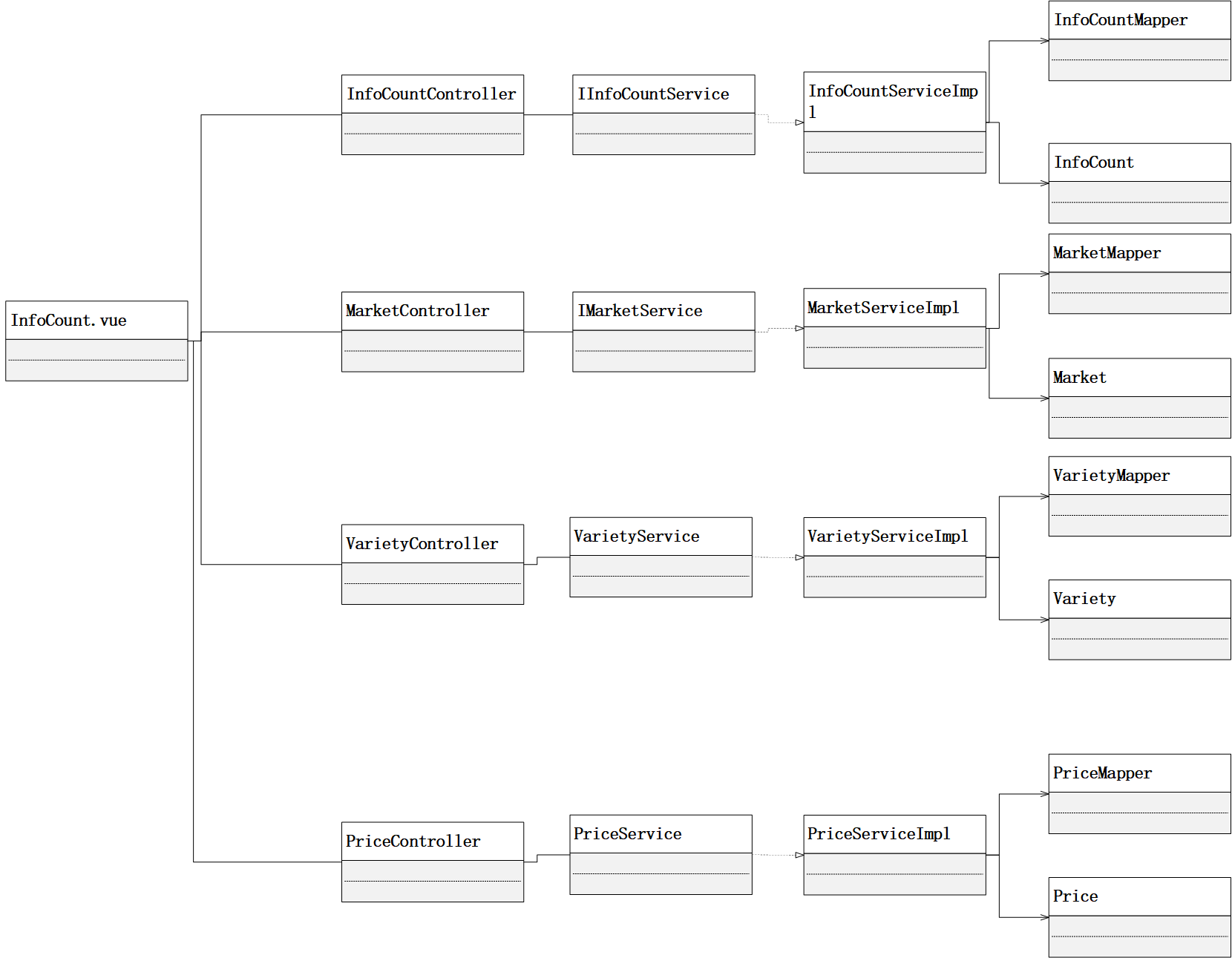
### **2、抓取量查**询

#### 2.1、功能汇总

该功能用户通过输入查询条件包括：区域信息、市场信息、品类信息、品种信息、起止时间信息，提交到服务端进行信息检索，返回查询结果。

|  |  |
| --- | --- |
| 功能 | 描述 |
| 选择查询市场 | 用户在浏览器界面选择查询条件时，根据选择的区域，更改市场下拉列表框的内容 |
| 选择查询品种 | 用户在浏览器界面选择查询条件时，根据选择的品类，更改品种下拉列表框的内容 |
| 抓取量查询 | 用户输入查询条件，向后台查询数据统计情况，前端得到返回信息，并显示 |

#### 2.2、类图设计



#### 2.3、类的详细设计描述

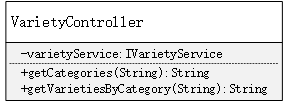
##### 1.3.1、MarketController描述



详细描述：

|  |
| --- |
| MarketController  为用户输入区域和市场提供下拉列表内容 |
| 1.   成员变量  private IMarketService marketService 市场查询服务接口 |
| 2.   public String getRegions() throws Exception  返回包含区域列表数据的Json字符串 |
| 3.   public String getMarketsByRegion(String region) throws Exception  根据输入的区域名返回包含该区域所有市场列表数据的Json字符串 |

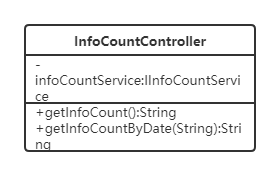
##### 1.3.2、VarietyController描述



详细描述：

|  |
| --- |
| VarietyController  为用户输入品类和品种提供下拉列表内容 |
| 1.   成员变量  private IVarietyService varietyService 市场查询服务接口 |
| 2.   public String getCategories() throws Exception  返回包含品类列表数据的Json字符串 |
| 3.   public String getVarietiesByCategory(String category) throws Exception  根据输入的品类名返回包含该品类所有品种列表数据的Json字符串 |

##### 1.3.3、 InfoCountController描述

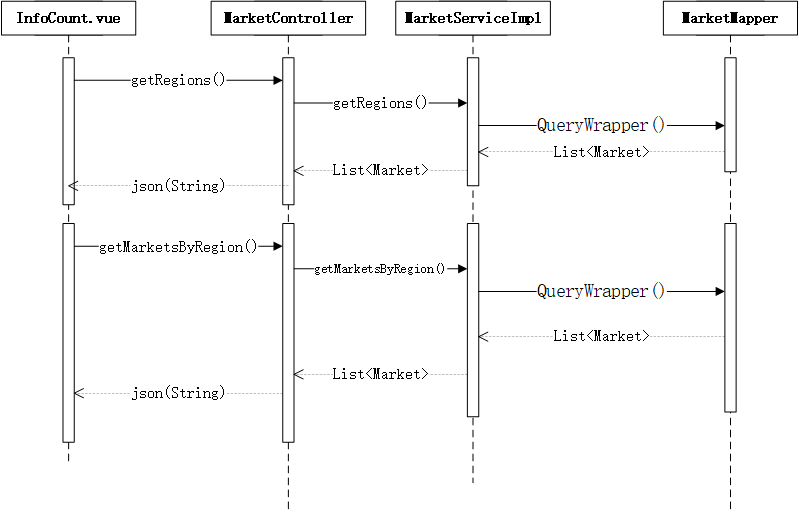


详细描述

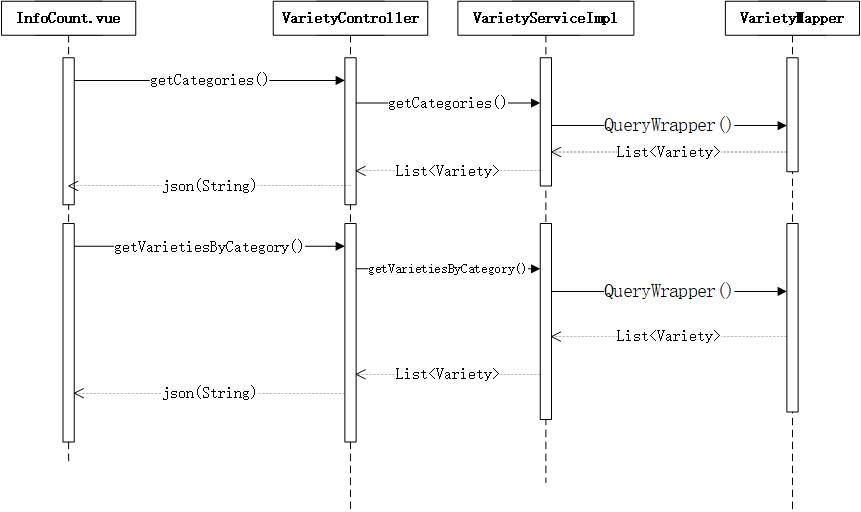
|  |
| --- |
| InfoCountController  根据用户输入信息执行查询，返回结果 |
| 1.   成员变量  private IInfoCountService infoCountService 市场查询服务接口 |
| 2.   public int getCountByDate(String date) throws Exception  返回包含执行查询所得结果数据的总量 |
| 3. public InfoCount getInfoCount(String)  返回某日的所有数据统计信息 |

#### 2.4、功能实现

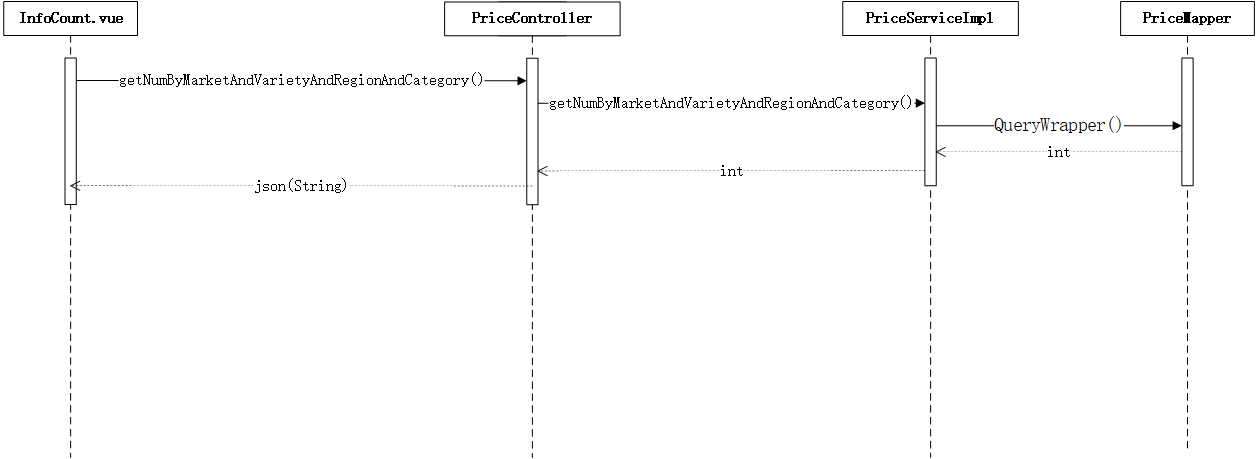
1. 选择查询市场



1. 选择查询品种



1. 抓取量查询



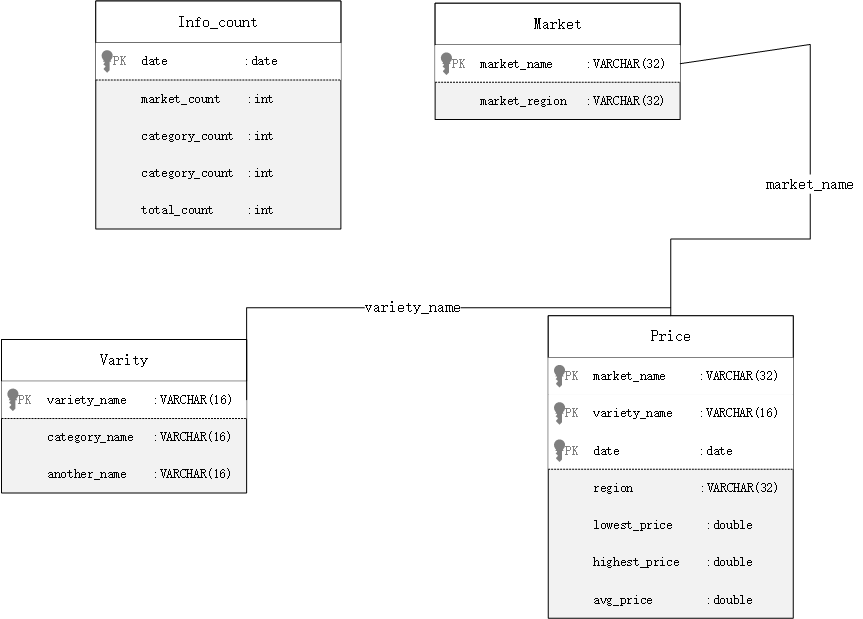
## 五. 接口文档说明

swagger-ui访问地址：[http://139.155.228.86:8083/swagger-ui.html#/](http://139.155.228.86:8083/swagger-ui.html" \l "/" \t "_blank)

接口说明文档doc： 接口文档说明.doc

第六部分 数据库设计

## 一、数据库整体结构图



## 二、数据库表格清单

|  |  |  |
| --- | --- | --- |
| 序号 | 表名 | 注释 |
| 1 | Market | 市场所属区域表 |
| 2 | Price | 价格表 |
| 3 | Variety | 品种类型表 |
| 4 | Info\_count | 数据统计表 |

### 1、Market表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 注释 |
| 1 | market\_name | VARCHAR(32) | 批发市场名 |
| 2 | market\_region | VARCHAR(32） | 批发市场所在地 |

### 2、Price表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 注释 |
| 1 | region | VARCHAR(32) | 批发市场所在地 |
| 2 | market\_name | VARCHAR(32) | 批发市场名 |
| 3 | variety\_name | VARCHAR(13) | 品种名 |
| 4 | lowest\_price | double(8) | 最低价格 |
| 5 | highest\_price | double(8) | 最高价格 |
| 6 | avg\_price | double(8) | 平均价格 |
| 7 | date | date | 日期 |

### 3、Variety表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 注释 |
| 1 | variety\_name | VARCHAR(16) | 品种名 |
| 2 | category\_name | VARCHAR(16) | 品类名 |
| 3 | another\_name | VARCHAR(16) | 别名 |

### 4、Info\_count表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 列名 | 数据类型 | 注释 |
| 1 | date | VARCHAR(16) | 日期r |
| 2 | market\_count | int | 市场总数 |
| 3 | category\_count | int | 品类总数 |
| 4 | variety\_count | int | 品种总数 |
| 5 | total\_count | int | 数据总数 |

### 5、数据库外键清单

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 外键名称 | 父表 | 父键列 | 子表 | 外键列 | 关系 | 说明 |
| market\_name | Price | market\_name | Market | market\_name | 1..\* | 一个批发市场对应一个区域，一个区域下可有多个批发市场 |
| variety\_name | Price | varity\_name | Varity | varity\_name | 1..\* | 一个品种只属于一个品类，一个品类可有多个品种 |

### 6、数据库建表sql语句

/\*

Navicat Premium Data Transfer

Source Server : 139.155.228.86

Source Server Type : MySQL

Source Server Version : 80019

Source Host : 139.155.228.86:3306

Source Schema : agriculture\_db

Target Server Type : MySQL

Target Server Version : 80019

File Encoding : 65001

Date: 08/08/2020 09:13:43

\*/

SET NAMES utf8mb4;

SET FOREIGN\_KEY\_CHECKS = 0;

-- ----------------------------

-- Table structure for info\_count

-- ----------------------------

DROP TABLE IF EXISTS `info\_count`;

CREATE TABLE `info\_count` (

`date` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

`market\_count` int(0) NULL DEFAULT NULL,

`category\_count` int(0) NULL DEFAULT NULL,

`variety\_count` int(0) NULL DEFAULT NULL,

`total\_count` int(0) NULL DEFAULT NULL

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

-- ----------------------------

-- Records of info\_count

-- ----------------------------

INSERT INTO `info\_count` VALUES ('2020-08-04', 60, 40, 30, 1000);

INSERT INTO `info\_count` VALUES ('2017-07-20', 1, 1, 1, 1);

-- ----------------------------

-- Table structure for market

-- ----------------------------

DROP TABLE IF EXISTS `market`;

CREATE TABLE `market` (

`market\_name` varchar(16) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

`market\_region` varchar(64) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

-- ----------------------------

-- Table structure for price

-- ----------------------------

DROP TABLE IF EXISTS `price`;

CREATE TABLE `price` (

`region` varchar(16) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`market\_name` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`variety\_name` varchar(16) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL,

`highest\_price` double(16, 0) NULL DEFAULT 0,

`lowest\_price` double(16, 0) NULL DEFAULT 0,

`avg\_price` double(16, 0) NULL DEFAULT 0,

`date` date NOT NULL

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci ROW\_FORMAT = Dynamic PARTITION BY KEY (`market\_name`,`date`)

PARTITIONS 127

(PARTITION `p0` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p1` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p10` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p100` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p101` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p102` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p103` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p104` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p105` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p106` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p107` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p108` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p109` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p11` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p110` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p111` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p112` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p113` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p114` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p115` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p116` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p117` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p118` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p119` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p12` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p120` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p121` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p122` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p123` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p124` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p125` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p126` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p13` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p14` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p15` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p16` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p17` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p18` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p19` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p2` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p20` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p21` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p22` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p23` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p24` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p25` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p26` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p27` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p28` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p29` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p3` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p30` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p31` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p32` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p33` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p34` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p35` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p36` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p37` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p38` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p39` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p4` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p40` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p41` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p42` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p43` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p44` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p45` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p46` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p47` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p48` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p49` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p5` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p50` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p51` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p52` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p53` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p54` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p55` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p56` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p57` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p58` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p59` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p6` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p60` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p61` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p62` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p63` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p64` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p65` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p66` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p67` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p68` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p69` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p7` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p70` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p71` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p72` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p73` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p74` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p75` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p76` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p77` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p78` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p79` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p8` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p80` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p81` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p82` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p83` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p84` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p85` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p86` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p87` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p88` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p89` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p9` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p90` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p91` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p92` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p93` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p94` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p95` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p96` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p97` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p98` MAX\_ROWS = 0 MIN\_ROWS = 0 ,

PARTITION `p99` MAX\_ROWS = 0 MIN\_ROWS = 0 )

;

-- ----------------------------

-- Table structure for variety

-- ----------------------------

DROP TABLE IF EXISTS `variety`;

CREATE TABLE `variety` (

`variety\_name` varchar(16) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

`category\_name` varchar(64) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL,

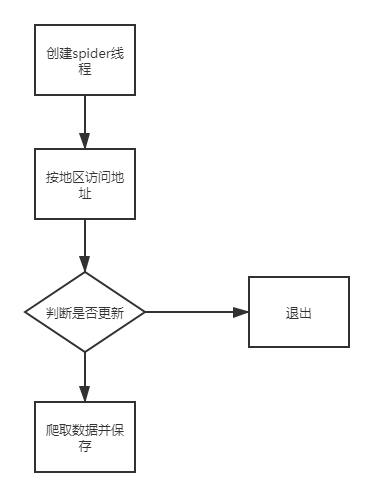
`another\_name` varchar(16) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL

) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;

SET FOREIGN\_KEY\_CHECKS = 1;

第七部分 数据爬取设计

## 一、数据爬取流程



## 二、Spider爬取数据

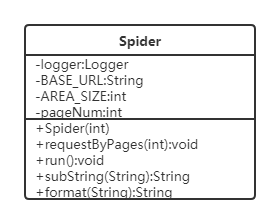
### 1.爬取数据

爬取数据的流程主要是访问根据地址访问网站上的内容再根据标签内容从返回值中提取出需要的字段保存到log文件中，其中对日期类型要做额外处理来与数据库的date类型对接，log文件保存在flume监听的目录下用于保留到数据仓库中。

### 2.提高效率

为了提高数据的爬取效率选择多线程爬取，每个线程负责爬取一个城市最新的数据，由于城市的数量较多而计算机处理线程的生命周期时需要消耗大量的内存，因此选取使用进程池对线程进行复用处理以减少内存开销，由于数据爬取为每天定时执行的短期任务，因此选取newCachedThredfPool线程池根据需要动态创建线程，尽可能的减少爬取所需的时间，提高爬虫的效率。

## 三、程序类图设计

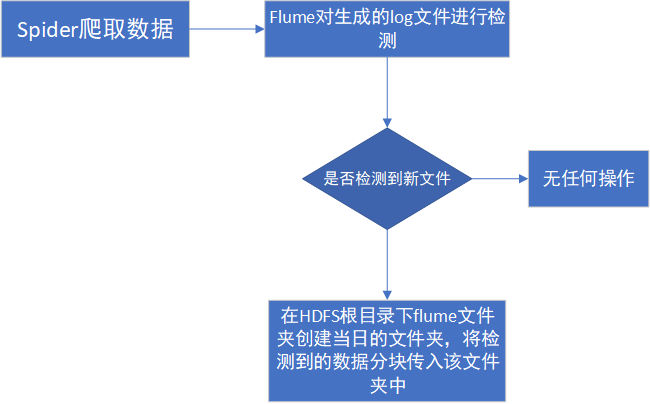


详细说明

|  |
| --- |
| Spider  爬取数据并保存 |
| 1.logger  日志采集实体，采集并将日志保存在指定目录下 |
| 2.BASE\_URL  访问页面的地址 |
| 3AREA\_SIZE  每个地区的目录页数 |
| 4.pageNum  爬取的起始页数 |
| 5.Spider(int)  构造方法，设定pageNum |
| 6.requestByPages()  爬取信息并保存 |
| 7.run()  线程的执行方法，执行requestByPages() |
| 8.subString(String)  匹配字符串 |
| 9.format（String）  将日期转化成标准格式 |

第八部分 数据处理设计

## 一、数据处理流程



## 二、Spark对数据进行处理

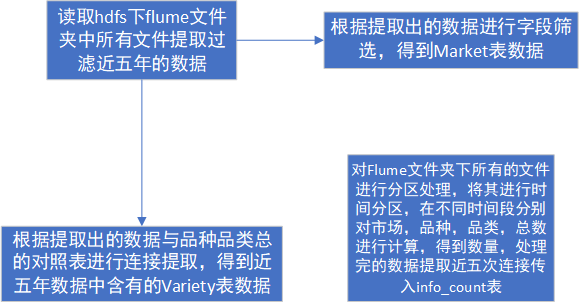
### 1.数据读取

由于本次使用的数据文件形式是log，因为使用csv文件可能出现hdfs自动分区出现读取错误。所以本次直接使用读取textFile形式。将读取到的数据进行预处理成为一个有标识字段的DataFrame类型。

### 2.数据处理

我们对数据进行处理主要为了获取InfoCount表，提取Market表和Variety表，对数据进行提取过滤。

所以步骤分为以下程序：



以上是第一次处理时需要做的事，之后每天只需要对mysql中Price表进行插入，对Market和Info\_count表进行更新即可。

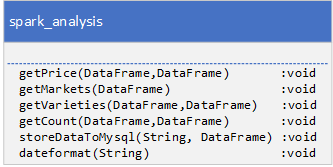
每过半年即执行一次数据清理，将过时的数据进行删除。即重复执行以上所有操作即可。（数据量只有百万级所以可以进行这样操作，若数据过亿级需要优化删除操作）

### 3.数据存储

数据存储方式使用append，目的在于不破坏数据库的表结构，对于第一次需要对数据库中数据进行清理然后append，之后每天执行append即可。

### 4.分析程序类图设计

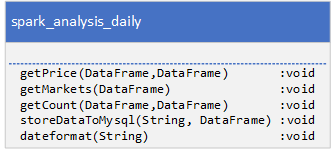
#### 4.1 spark\_analysis说明



详细说明

|  |
| --- |
| spark\_analysis  执行数据筛洗，写表初始化 |
| 1. getPrice(source: DataFrame,variety:DataFrame)  只保留近五年的数据，并写入mysql中，执行关联方法有getMarkets,getVarieties,storeDataToMysql |
| 2. getMarkets(source: DataFrame)  筛选近五年内所有的市场和区域对照数据并写入Market表中，关联方法有storeDataToMysql |
| 3. getVarieties(sourceDF: DataFrame,varietyDF: DataFrame)  筛选近五年内所有的品类和品种对照数据并写入Variety表中，关联方法有storeDataToMysql |
| 4. getCount(sourceDF: DataFrame,varietyDF:DataFrame)  根据日期分别计算近五次的市场总数，品类种数，品类总数，价格总数数据并写入Info\_count表中，关联方法有storeDataToMysql |
| 5. storeDataToMysql(tableNameInMysql: String, result: DataFrame)  根据表名和数据进行数据库中数据写入，先清洗后append |
| 6. dateformat(date:Date)  将日期的格式改为yyyy-MM-dd (eg: 2020-08-02) |

#### 4.2 spark\_analysis\_daily说明



详细说明

|  |
| --- |
| spark\_analysis\_daily  执行数据筛洗，写表初始化 |
| 1. getPrice(source: DataFrame,variety:DataFrame)  读取当日的数据并写入mysql，关联方法有storeDataToMysql |
| 2. getMarkets(source: DataFrame)  筛选近五年内所有的市场和区域对照数据并写入Market表中(包含当日数据)，关联方法有storeDataToMysql |
| 3. getVarieties(sourceDF: DataFrame,varietyDF: DataFrame)  筛选近五年内所有的品类和品种对照数据并写入Variety表中（包含当日数据），关联方法有storeDataToMysql |
| 4. getCount(sourceDF: DataFrame,varietyDF:DataFrame)  根据日期分别计算近五次的市场总数，品类种数，品类总数，价格总数数据并写入Info\_count表中，关联方法有storeDataToMysql |
| 5. storeDataToMysql(tableNameInMysql: String, result: DataFrame)  根据表名和数据进行数据库中数据写入append |
| 6. dateformat(date:Date)  将日期的格式改为yyyy-MM-dd (eg: 2020-08-02) |